Reconsideració dels models conceptual i lògic

Alberto Abelló Gamazo
Jaume Sistac i Planas

P01/11031/00006

Índex

Introducció 5

Objectius 6

1. Paranys de disseny 7
1.1. Ventall 7
1.2. Tall 8
1.3. Pèrdua d'afiliació 9
1.4. Aritat de les associacions 11
1.5. Semàntica de les classes 12

2. Consideracions en el pas a model lògic 13
2.1. Desaparició de taules corresponents a classes 13
2.2. Comprovacions d'instàncies 14
2.3. Atributs d'associacions 15
2.4. Restriccions d'integritat 16
2.4.1. Alternatives d'implementació 16
2.4.2. Substitució 17

3. Representació en el model lògic de la generalització/especialització 19

4. Representació d'atributs instanciats en diferents valors 21

5. Abraçades mortals de definició i de càrrega 23

6. Substituts de la clau primària 26

7. Freqüència de processos i volums de dades 28
7.1. Freqüència de processos 28
7.2. Volums de dades (fragmentació de taules) 28

8. Redundància de dades: duplicades i derivades 31
8.1. Duplicació de dades 31
8.2. Dades derivades 32

9. Històrics 34

Resum 36

Activitats 37

Exercicis d'autoavaluació 37

Solucionari 38

Glossari 38

Bibliografia 39

Introducció

En altres mòduls hem presentat el llenguatge de modelització UML i s'ha explicat com fer-lo servir a l'etapa de disseny per a obtenir el model conceptual d'una base de dades (BD). A més, també hem explicat com obtenim el conjunt de taules amb les seves columnes, claus primàries i claus foranes que tindrem a la nostra base de dades a partir d'aquest model conceptual expressat en UML.

El llenguatge UML i la seva transformació a model relacional s'han estudiat en el mòdul "Disseny conceptual i lògic de bases de dades" d'aquesta assignatura.

UML és la sigla d'*Unified Modeling Language*. Abreugem *base de dades* amb la sigla *BD*.

La modelització no és una activitat que es pugui sistematitzar, té un gran component de creativitat i depèn molt del punt de vista de cada persona (dues persones poden concebre la mateixa realitat de maneres diferents). No hi ha cap algorisme determinista que, atesa una part del món real que ens interessi, ens permeti d'obtenir-ne la conceptualització. Ni tan sols hi ha una conceptualització única d'una realitat.

Fins ara s'han donat només els elements que componen l'UML i el seu significat, juntament amb una sèrie de pautes que s'han de seguir per a arribar a tenir les taules de la base de dades relacional. En aquest mòdul veurem que fer un disseny no és tan simple com pot semblar a primer cop d'ull i que hi ha molts punts que cal tenir en compte quan dissenyem. Cadascun dels apartats del mòdul presenta un problema de disseny i dóna un conjunt de solucions possibles juntament amb els criteris que es poden seguir per a triar la solució que més convé a les nostres necessitats.

Penseu que no n'hi ha prou de tenir a l'abast els criteris necessaris per a fer un bon disseny. A dissenyar, se n'aprèn dissenyant, i no sols llegint. Haureu d'entrenar la vostra capacitat d'abstracció i el vostre judici per a prendre la millor decisió a l'hora de triar una representació per a la informació que us interessi guardar.

Objectius

Aquest mòdul mostra alguns problemes de disseny, juntament amb un conjunt de solucions possibles i criteris que s'han de tenir en compte per tal de triar-ne les millors. A mesura que us introduïu en el mòdul, assolireu els objectius següents:

1. Despertar l'esperit crític davant d'un disseny (no existeix "la" representació de la informació, sinó que n'hi ha moltes i hem de triar la millor).

2. Reconèixer un conjunt de problemes de disseny relativament habituals.

3. Plantejar solucions alternatives a un problema de disseny.

4. Disposar d'alguns criteris que puguin guiar la tria d'una solució a un problema en un cas concret.

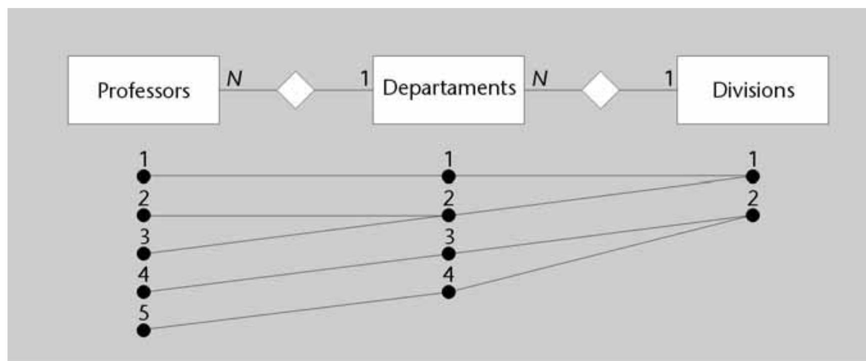
1. Paranys de disseny

En aquest apartat veurem cinc paranys en què podem caure quan dissenyem. Aquests paranys estan relacionats amb conceptualitzacions errònies de la realitat que s'acaben reflectint en la modelització. És molt important no caure-hi perquè trobaríem greus problemes a l'hora de posar en marxa la nostra base de dades*.

* Per exemple, consultes que no podem resoldre.

Organització de professors a la Universitat de Barcelona

Per a il·lustrar els dos primers paranys utilitzarem com a exemple l'organització del professorat a la Universitat de Barcelona.



Tal com mostra la figura, els professors s'agrupen en departaments que componen les divisions. La seva traducció al model relacional ens donaria una taula per cada classe, amb dues claus foranes de *Professors* a *Departaments* i de *Departaments* a *Divisions*. Simplement amb dues operacions de combinació (*join*) podem obtenir també a quina divisió pertany cada professor:

Combinacions

S'entén per *combinació* l'operació `join` del model Relacional.

```
SELECT nom_div, nom_prof
FROM Professors p,  Departaments e, Divisions i
WHERE p.clau_dept = e.clau AND e.clau_div = i.clau;
```

1.1. Ventall

Caure en el parany del ventall provoca que no puguem resoldre totes les consultes que voldríem. Es pot donar sempre que trobem dues associacions amb multiplicitat 1:*N* "encadenades". És a dir, quan tenim una classe *A* que està associada amb *B* i, a la seva vegada, *B* està associada amb *C*. En aquest cas, fàcilment podem interpretar que, per transitivitat, *A* està associada amb *C*. Sembla clar que no cal representar totes tres associacions, ja que una es pot deduir a partir de les altres.

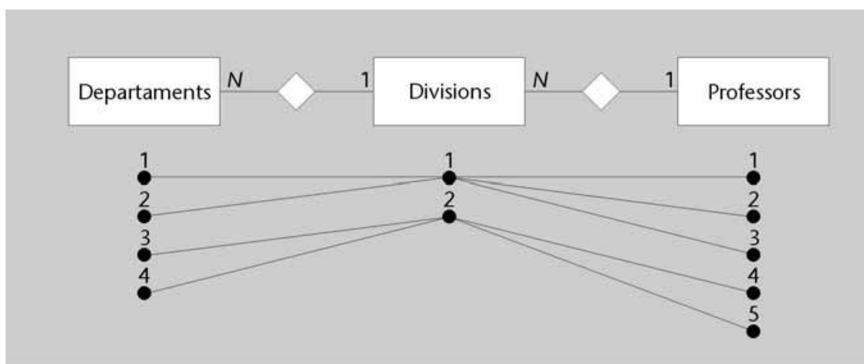
Arribats en aquest punt és quan podem caure al parany de triar les dues associacions errònies, cosa que ens portaria a perdre informació. És molt important d'escollir les dues associacions realment significatives. Si errem en triar-

les, podem trobar que hi ha consultes que no podem contestar amb la nostra mal dissenyada base de dades. Hem de parar molta atenció i triar les dues que realment corresponen al que volem modelitzar.

D'aquest parany, en diem **parany de ventall** perquè en dibuixar les instàncies de les classes, entre aquestes apareix una mena de ventall format per les associacions. Aquest ventall fa que no puguem saber com s'associen les instàncies de les classes que queden als extrems.

Exemple de parany de ventall

En la figura següent tenim un disseny erroni de l'organització de professors a la Universitat de Barcelona:



Aquí podem veure un exemple de ventall. Els professors pertanyen a divisions que estan compostes per departaments. Aquest disseny dóna lloc a una clau forana de ***Professors*** a ***Divisions*** i una altra de ***Departaments*** també a ***Divisions***. Amb aquest disseny, no podríem contestar a la pregunta: "A quin departament pertany cada professor?". No tenim cap relació directa entre ***Departaments*** i ***Professors***, i si tractem de fer combinacions (***joins***) per a respondre a aquesta pregunta, arribem a la conclusió errònia que un professor pertany a molts departaments:

```
SELECT nom_dept, nom_prof
FROM Departaments e, Divisions i, Professors p
WHERE e.clau_div = i.clau AND i.clau = p.clau_div;
```

Aquesta consulta ens diria que el professor 1 pertany als departaments 1 i 2, cosa que és impossible, perquè un professor només pot pertànyer a un departament.

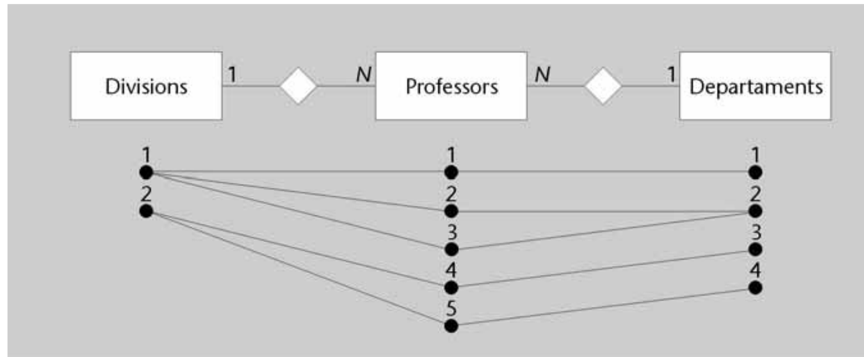
1.2. Tall

Aquest segon parany és molt semblant a l'anterior, en el sentit que també es produeix quan trobem tres associacions on una es dedueix de les altres dues i interpretem malament quines són les dues realment significatives. Però ara el problema que trobem en caure en aquest parany és una mica més subtil que el del subapartat anterior. En general, no hi ha cap pèrdua d'informació. Sembla que puguem resoldre totes les consultes que ens interessen. Però sí que n'hi ha, de pèrdua d'informació, quan esborrem instàncies de la classe que queda al mig.

D'aquest parany en diem **parany de tall**, perquè si esborrem instàncies de la classe central, les associacions entre instàncies de les classes que hi ha als extrems poden quedar tallades.

Exemple de parany de tall

Tornem a l'exemple de l'organització del professorat a la Universitat de Barcelona. Podríem caure en el parany de modelitzar-lo de manera que, d'una banda, un professor pertany a un departament, i de l'altra, pertany a una divisió, com es pot veure a la figura:



Aquesta representació dóna lloc a dues claus foranes a la taula *Professors*: una apunta a *Divisions* i l'altra, a *Departaments*. Aleshores, pensem en el cas hipotètic que un departament no té cap professor o que esborrem tota la informació dels professors d'un departament. En aquest cas, no podríem respondre la pregunta: "A quina divisió pertany aquest departament?" Si tractéssim de fer la combinació (*join*) entre *Divisions* i *Departaments*, trobaríem que no hi ha cap fila a la taula *Professors* que lligui el departament amb la seva divisió:

```
SELECT nom_dept, nom_div
FROM Divisions i, Professors p,  Departaments e
WHERE i.clau = p.clau_div AND p.clau_dept = e.clau;
```

El resultat d'aquesta consulta és buit per a tots els departaments que no tinguin cap professor assignat.

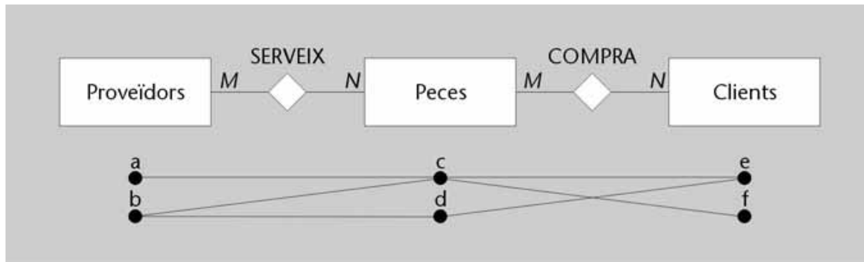
1.3. Pèrdua d'afiliació

Encara hi ha un altre parany que podem trobar quan tenim tres classes relacionades amb dues associacions. En aquest cas, les associacions haurien de tenir multiplicitat N:M per a caure-hi. D'aquesta manera, si existís una relació directa entre les dues classes als extrems, no podríem registrar-la de cap manera.

El parany de pèrdua d'afiliació es podria veure com un parany de doble ventall. Una instància a la classe central pot estar relacionada amb tot un ventall d'instàncies a cadascuna de les altres dues classes. Així, l'únic que podríem dir és que una instància a un dels ventalls està relacionada amb algunes (com a mínim una, però potser totes) instàncies a l'altre ventall. És impossible concretar amb quines.

Exemple de pèrdua d'afiliació

Imaginem un magatzem en què temporalment guardem peces provinents de diferents proveïdors per a distribuir-les als nostres clients. Tal com veiem a la figura, una peça pot venir de molts proveïdors i un proveïdor ens serveix peces diferents. Per la seva banda, cada client ens pot comprar diferents tipus de peces i nosaltres podem vendre un determinat tipus de peça a diferents clients.



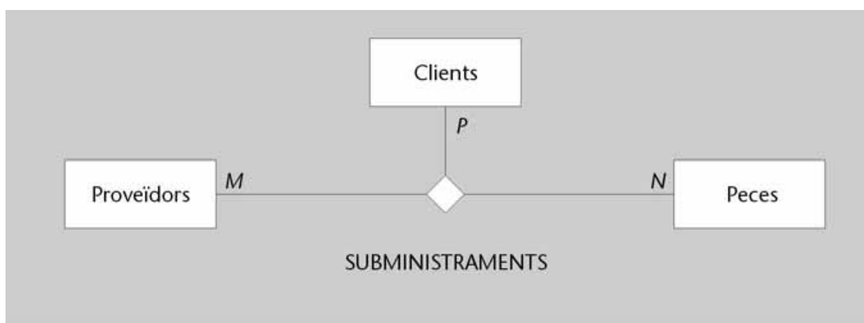
En el model relacional, per cada associació tindríem una taula amb una clau forana cap a cadascuna de les taules corresponents a les classes que relaciona. Amb aquest disseny no podem respondre la pregunta: "De quin proveïdor provenien les peces que hem venut a cada client?". No ho podem saber de cap manera, perquè en fer la combinació (*join*) d'aquestes dues taules de les associacions ***serveix*** i ***compra***, obtindrem en realitat totes les possibles parelles que hi ha de proveïdor i client per a cada peça.

```
SELECT s.proveidor, c.peça, c.client
FROM serveix s, compra c
WHERE s.peça = c.peça;
```

No sempre n'hi ha prou amb dues associacions binàries per a representar la informació que ens convé. En aquests casos, hauríem de modelitzar la relació amb una associació ternària.

Solució a la pèrdua d'afiliació

En el cas dels subministraments de peces, cal un disseny com el que veiem en aquesta figura.



Aquí trobem una associació ternària entre ***Proveïdors***, ***Peces*** i ***Clients***. Això dóna lloc a una única taula amb tres claus foranes (una cap a cadascuna de les taules de les classes), de manera que podem enregistrar cadascun dels subministraments per separat. Així sabem de quin proveïdor hem rebut cada peça i a quin client l'hem enviada. Com que ara només tenim una taula, no cal fer la combinació (*join*) i evitem el problema anterior.

```
SELECT proveidor, peça, client
FROM subministraments;
```

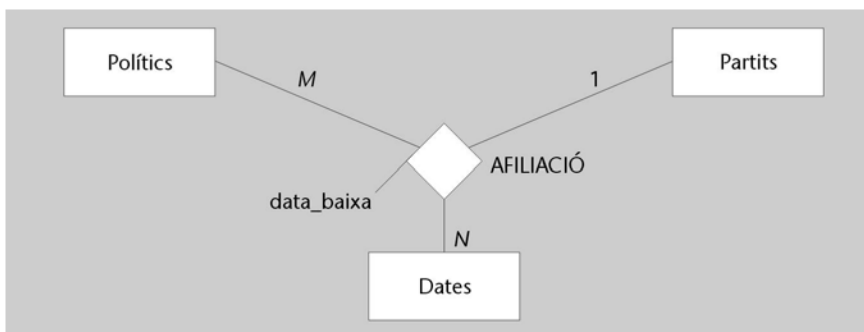
1.4. Aritat de les associacions

L'aritat de les associacions també pot generar errors en el disseny. Hem d'anar amb molt de compte per veure quantes classes participen realment en cada associació.

En general, hem de parar atenció a les associacions que involucren més de dues classes i alguna té multiplicitat 1. En aquests casos, és possible que aquesta classe estigui realment relacionada només amb una de les altres classes i que no hagi de participar en l'associació. Ens hem de fixar en la raó per la qual hi ha l'1.

Exemples de paranys d'aritat

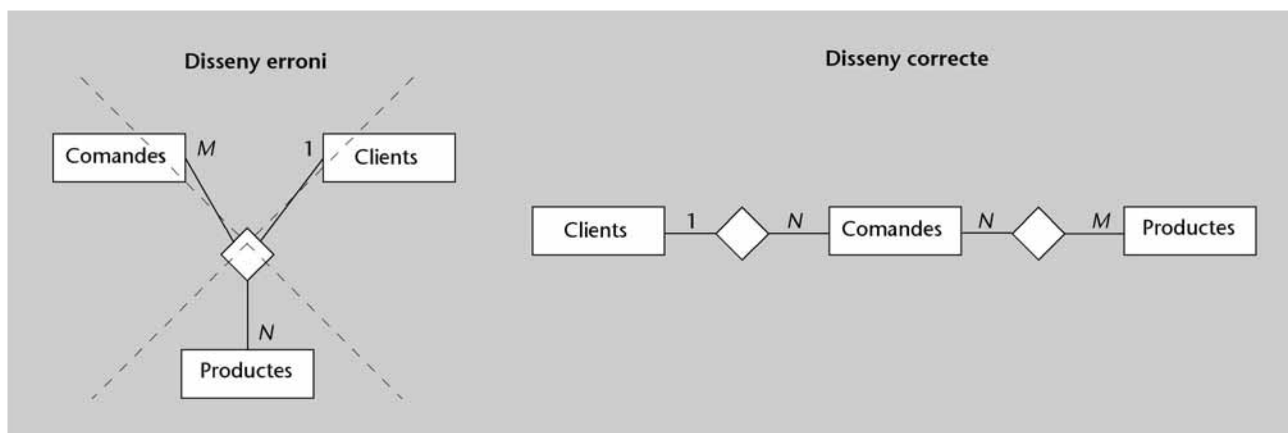
a) En la figura següent veiem representades les afiliacions a partits polítics:



Òbviament, un partit polític determinat no està relacionat directament amb les dates. A més tampoc no ho pot estar amb els polítics, ja que ni neixen pertanyent a cap, ni necessàriament pertanyen a un partit per tota la vida. Si analitzem la multiplicitat, veiem que l'1 apareix perquè en una data determinada, un polític només es pot afiliar a un partit concret. És a dir, un polític i una data determinen unívocament el partit en el qual hi ha hagut l'afiliació (hi ha una dependència funcional plena de polítics i dates cap a partit).

Les dependències funcionals es veuen en el subapartat 1.2 del mòdul "Disseny conceptual i lògic de bases de dades" d'aquesta assignatura.

b) En l'exemple següent trobem una associació ternària per a la qual no trobem nom (mal símptoma).



Si intentem traduir-ho al model relacional, ens adonarem que no surt ni tan sols en segona forma normal. Obtenim una taula amb tres claus foranes cap a *Comandes*, *Productes* i *Clients*, on les dues primeres formen la clau primària. Així, el client depèn funcionalment de la comanda i del producte, però aquesta dependència no és plena. Només la comanda ja identifica el client. El producte no cal, ja que tots els productes dins una

comanda són per al mateix client. Per tant, les *Comandes* estan relacionades directament amb *Clients*, sense involucrar *Productes* per a res.

En cas que trobem que l'associació té l'aritat que pensàvem, encara hem de veure si aquesta classe amb multiplicitat 1 és realment una classe, o la podem tractar com un simple atribut de l'associació. Observem que en fer la traducció al model relacional, la clau primària de la taula està composta només per les claus primàries de les classes amb multiplicitat més gran que 1.

Així, tant si considerem l'existència d'una classe amb multiplicitat 1, com si considerem que simplement tenim un atribut de l'associació, obtindrem el mateix model lògic. En aquest cas, hem de matisar la importància de les dades. Si trobem que la classe en qüestió no té cap atribut ni cap altra associació, fóra millor de representar-la només com un atribut de l'associació.

1.5. Semàntica de les classes

Un altre punt important de considerar és el significat real dels elements del nostre disseny. L'orientació a l'objecte (OO) posa molt d'èmfasi en la correspondència dels elements del nostre model amb entitats o conceptes reals. Per tant, s'ha d'evitar barrejar atributs d'entitats diferents dins una mateixa classe.

Abreugem *orientació a l'objecte* amb la sigla *OO*.

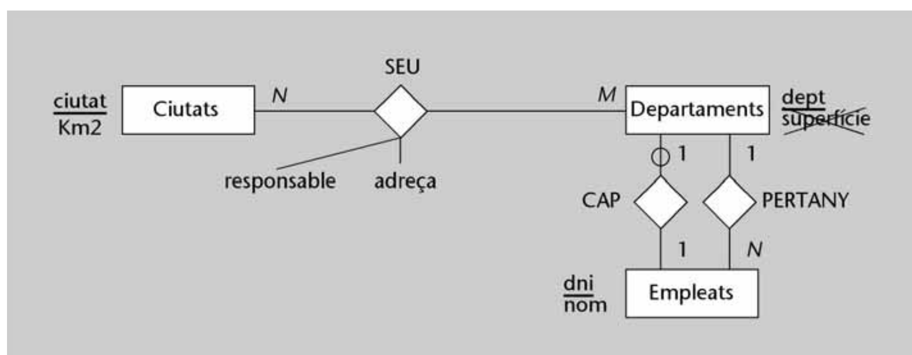
Cal parar una especial atenció al significat real que té el que es vol representar i no deixar-nos portar pel que pugui semblar a primer cop d'ull. Hem d'entendre el significat real de cada classe per a posar-hi només els atributs que realment li pertoquen.

No us deixeu enganyar!

Hi ha casos en què el nom d'una classe ens pot enganyar sobre la genericitat d'aquesta (com és de genèrica). Si la classe representa un concepte abstracte o organitzatiu, no pot ser que tingui atributs que facin referència a qüestions físiques i a l'inrevés.

Exemple de parany en la semàntica de les classes

En aquesta figura, veiem que un departament està situat a diferents ciutats i té empleats, dels quals un n'és el cap. Podríem tenir un atribut *superfície* a la classe *Departaments* que ens donés els metres quadrats que té cada departament? La resposta és "no", ja que seria més apropiat tenir aquest atribut per cada seu del departament a les diferents ciutats.



En l'exemple, *Departaments* és una classe genèrica que representa una agrupació organitzativa de l'empresa, mentre que la superfície és un atribut físic molt més lligat a les seus del departament. *Departaments* podria tenir, per exemple, un atribut que indiqués la superfície que en teoria correspondria a cada departament segons el pla estratègic de l'empresa. Però els metres quadrats que ocupa en realitat és un atribut de cadascuna de les seus.

2. Consideracions en el pas a model lògic

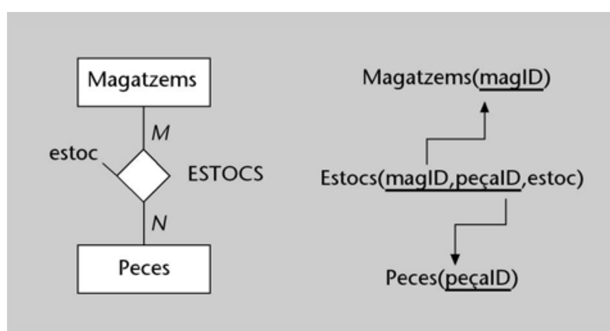
Un cop tenim el model conceptual de la nostra BD, en UML, la seva traducció al model relacional clàssic és relativament senzilla. Això no obstant, cal no oblidar que fóra bo aturar-se una estona a reflexionar sobre alguns punts. En aquest apartat veurem algunes petites consideracions que hem de tenir en compte.

2.1. Desaparició de taules corresponents a classes

Si una classe apareix en el nostre model conceptual és perquè realment ens interessa i, per tant, ja està bé que hi sigui. No s'ha de treure pas de l'esquema conceptual. Però ens hem de plantejar si té prou importància com per a donar lloc a una taula en la nostra base de dades. Té sentit guardar una taula en què totes les columnes formen part de la clau primària i només serveixen perquè s'hi faci referència des d'una altra taula? Si una classe del nostre model conceptual no té cap atribut que no formi part de la clau primària, hem de veure si el fet de guardar explícitament les instàncies d'aquesta classe ens és útil o no (podria ser molt important per a comprovar la integritat referencial).

Exemple de classes sense atributs que cal conservar en el model lògic

Si volem guardar quantes peces hi ha en cada magatzem, tal com podem veure a la figura següent, podríem trobar que hi ha dues classes sense atributs (*Magatzems* i *Peces*).



Ara bé, cal parar atenció al fet que les taules corresponents ens serviran per a comprovar la integritat referencial. Podem tenir milers de peces identificades per codis de barres o números que ningú no coneix i molt fàcils de confondre. És imprescindible de disposar de la clau forana per tal d'evitar errors en la introducció de dades.

En el cas dels magatzems, pot semblar que això no és gaire important de fer. Si només tenim quatre magatzems, qui s'equivocarà i posarà un nom que no existeix? A més, encara que algú s'equivoqués, no trigaríem gaire a adonar-nos-en. Tot i això, hauríem de tenir una bona raó per a no posar la clau forana. Tot i que en aquest cas, la probabilitat de tenir problemes d'integritat sigui mínima, sempre és millor deixar la clau forana perquè fa que la possibilitat simplement no existeixi.

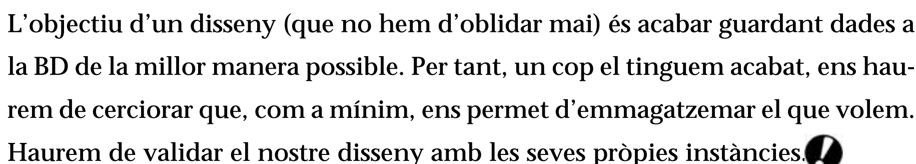
> En general, no és mai bo deixar de guardar cap informació, per negligible que pugui semblar. Les claus foranes són molt importants per a mantenir la integritat de la nostra base de dades.

Potser el cas que podríem considerar més clar que no cal guardar una taula són les dates. Qualsevol sistema de gestió de bases de bades (SGBD) ens ofereix un tipus de dades *Date*, amb el seu domini propi. Per tant, sembla que no cal que tinguem una taula que mostri quines són les dates vàlides, perquè el sistema mateix farà les comprovacions necessàries. Això no obstant, cal sospesar què fem atès que, si no tenim aquesta taula de dates, on guardarem la informació sobre les festes que són recuperables i les que no, a la nostra empresa? En cas de tenir una BD d'una universitat, on guardarem quins són els dies lectius, d'exàmens, de matriculació, etc.?

Abreugem *sistema de gestió de bases de dades* amb la sigla SGBD.

> Només és possible la supressió d'una taula si no guarda cap atribut i les possibles comprovacions d'integritat es fan d'alguna altra manera.

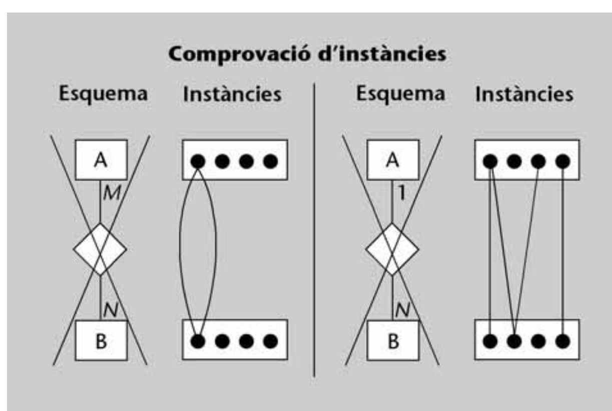
2.2. Comprovacions d'instàncies



L'objectiu d'un disseny (que no hem d'oblidar mai) és acabar guardant dades a la BD de la millor manera possible. Per tant, un cop el tinguem acabat, ens haurem de cerciorar que, com a mínim, ens permet d'emmagatzemar el que volem. Haurem de validar el nostre disseny amb les seves pròpies instàncies.

Validació de les multiplicitats de les associacions

Com a casos especialment importants que convé validar, podem ressaltar les multiplicitats de les associacions.



En la part esquerra de la figura anterior podem veure una suposada associació *N*:*M* binària instanciada dos cops pels mateixos elements. Si pensem en la traducció que això tindrà al model relacional, ens adonarem que hi haurà una violació de clau primària en la taula a què l'associació donarà lloc. Probablement, ho hauríem de modelitzar amb una associació ternària que involucrés una tercera classe.

La part dreta de la mateixa figura ens mostra una associació 1:*N* en què un element es relaciona amb molts, i molts es relacionen amb un mateix element. En la traducció de l'associació 1:*N* al model relacional no podrem representar això. Si realment ho volguéssim fer, la multiplicitat hauria de ser *N*:*M*.

Amb això no volem pas dir que per a dissenyar s'hagin de mirar les instàncies. Aquest podria ser un error encara més greu. Allò realment important és la semàntica, el significat de les dades, i no pas les instàncies. Només pel fet que tinguem o deixem de tenir unes instàncies en un moment determinat, no hem de pensar que allò passarà sempre.

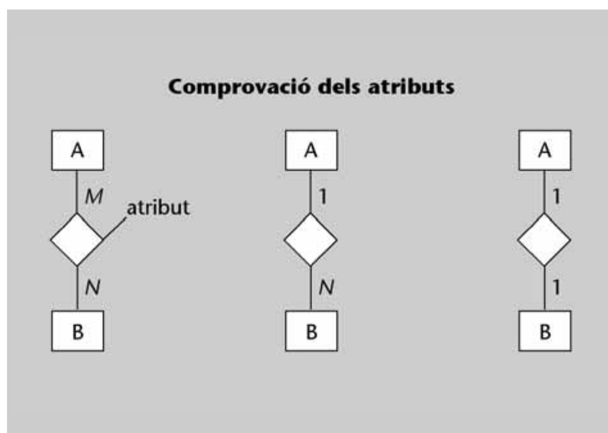
Exemple d'interpretació errònia de les instàncies

Si modelitzem una BD d'alumnes i assignatures, del fet que en una sèrie d'assignatures no trobéssim cap suspès, mai no hauríem d'inferir que no hi pot haver estudiants suspesos. El significat de les notes l'hauria de donar el professor de cada assignatura o el sistema educatiu.

Les instàncies ens poden ajudar a trobar errades, però mai no han de dirigir el disseny. Això ho ha de fer la semàntica de les dades.

2.3. Atributs d'associacions

Un altre punt molt important que hem de considerar quan fem un disseny és de qui són els atributs. Hi ha casos en què trobarem que aquests pertanyen a classes, i d'altres en què pertanyen a les associacions.



Exemple d'associació *N*:*M* amb atributs

En una BD en què guardem *Alumnes* i *Assignatures*, a qui pertanyen les notes, als alumnes o a les assignatures? Un alumne en si mateix no té cap nota (no és un atribut de l'alumne), sinó que dependrà de l'assignatura. Una assignatura tampoc no té cap nota prefixada (les assignatures tampoc no tenen notes), sinó que dependrà de cada estudiant. Per tant, les notes no són ni dels estudiants ni de les assignatures, sinó de l'associació que hi ha entre aquests.

En general, tal com es representa a la figura anterior, les associacions *N*:*M* acostumen a tenir atributs, mentre que les que tenen multiplicitat 1:*N* o 1:1 acostumen a no tenir-ne. En cas de trobar-nos un d'aquests dos casos, ens hem d'assegurar que l'atribut és realment de l'associació. Si tenim una associació 1:*N*, és probable que l'atribut pertanyi a la classe que hi ha al costat *N*. Però si tenim una associació 1:1, l'atribut podria pertànyer a qualsevol de les dues classes. En traduir aquestes associacions a model lògic, veiem que una associació *N*:*M* dóna lloc a una taula que tindrà una columna per cadascun dels atributs, juntament amb les claus foranes cap a les dues taules de les classes. En canvi, les associacions 1:*N* i 1:1, és representen simplement amb una clau forana en una de les dues taules corresponents a les classes.

Si la multiplicitat de l'associació és 1:*N*, tant la clau forana com els possibles atributs de l'associació es guarden a la taula de la classe situada al costat *N*. Si l'associació té multiplicitat 1:1, haurem de triar en quina de les dues taules posem tant la clau forana com les columnes corresponents als possibles atributs de l'associació. Mirarem si algun dels dos costats de l'associació admet zeros i veurem també què passa amb els atributs en aquests casos, per tal d'intentar que no apareguin valors nuls a cap columna de la taula (ni a les que formen la clau forana ni a les que guardem els atributs de l'associació).

2.4. Restriccions d'integritat

Deixant de banda les facilitats de cerca, el gran avantatge d'utilitzar una BD informatitzada en comptes dels antics arxivadors amb papers és que es poden fer comprovacions automàtiques sobre les restriccions d'integritat (RI).

Abreugem *restriccions d'integritat* amb la sigla *RI*.

Tenir dades errònies pot ser fins i tot pitjor que no tenir res i les **restriccions d'integritat** ajuden a controlar l'existència d'errors. En el disseny, no se n'ha de negligir la importància, i sobretot, no les hem d'oblidar quan aquest disseny s'implementi sobre un SGBD.

2.4.1. Alternatives d'implementació

Els SGBD actuals ens proposen quatre alternatives per a implementar RI, entre les quals podem triar la que més ens convingui en cada cas:

1) A la sentència de creació de taules, podem incloure:

- Clau primària (*Primary Key*)
- Clau forana (*Foreign Key*)
- No admet valors repetits (*Unique*)
- No admet valors nuls (*Not null*)
- Comprovacions de condicions sobre columnes (*Check*)

2) L'SQL/PSM (*Persistent Stored Modules*) permet de definir disparadors (*triggers*), procediments (*procedures*), funcions (*functions*) i paquets (*packages*) mitjançant els quals es poden fer les comprovacions que calguin.

SQL/PSM en alguns SGBD

En l'Oracle, d'això se'n diu PL/SQL (*Programming Language SQL*), mentre que Informix en diu SPL (*Stored Programming Language*).

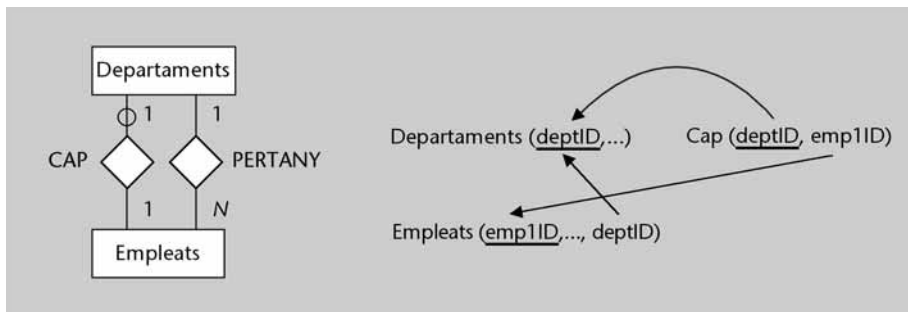
3) SQL Host permet la inclusió de sentències SQL directament dins programes en un llenguatge determinat (per exemple, el Java té l'SQLJ). Mitjançant un procés de precompilació de les aplicacions, aquestes sentències es tradueixen en crides a l'SGBD, que poden utilitzar-se per a comprovar RI.

4) Finalment, SQL/CLI (*Call Level Interface*) és una interfície estàndard que implementen els SGBD per a diferents llenguatges de programació, que facilita l'accés a la BD (per exemple, tenim JDBC per a Java). Es tracta de llibreries de funcions que podem cridar des de la nostra aplicació i que, en el procés d'acoblament (*link*) es traduirà en crides sobre l'SGBD corresponent. Això ens pot ser especialment útil ja que permet d'utilitzar diferents llibreries alhora per a accedir a diferents BD en SGBD diferents des de la mateixa especificació.

Vegeu l'explicació d'*SQL Host* i *SQL/CLI* en el mòdul "Programació amb SQL" de l'assignatura *Bases de dades II*.

Representació de restriccions d'integritat en un SGBD relacional

En l'exemple següent podem veure com algunes restriccions d'integritat es poden representar més fàcilment que d'altres en un SGBD relacional:



Principalment, distingirem dos tipus de restriccions d'integritat:

1) Reflectides en el model:

a) Per a fixar que un empleat pertany a un departament, i només un, podem indicar que hi ha una clau forana d'*Empleats* a *Departaments* i que no admet valors nuls.

b) Per a forçar que cada departament té un cap, i que només en té un (igual que abans), podem tenir que la clau forana de *Cap* a *Empleats* no admet valors nuls, amb la qual cosa garantim que si un departament té una fila a la taula *Cap*, un dels empleats serà el cap del departament. Un departament no podria tenir més d'un cap, perquè l'identificador del departament és la clau primària de *Cap*. Però cap d'aquestes dues RI no garanteix que cada departament tingui cap. Cal un programa amb SQL Host, SQL/CLI, o SQL/PSM que comprovi que a la taula *Cap* hi ha una fila per cadascun dels departaments a la taula *Departaments*.

c) Si, a més, volem imposar que un empleat no pot ser cap de més d'un departament, hauríem d'afegir que la clau forana de *Cap* a *Empleats* no admet valors repetits.

2) No reflectides en el model

a) Per a indicar que un departament pot tenir fins un màxim de vuit empleats, podríem utilitzar, per exemple, un disparador.

b) Si volem forçar que el cap d'un departament pertanyi al mateix departament, ho podríem fer també amb un disparador.

2.4.2. Substitució

Mantenir la consistència d'una BD és imprescindible. Això no obstant, s'ha de saber que les comprovacions d'RI alenteixen considerablement el funcionament del sistema. Tant és així que de vegades cal deslliurar l'SGBD d'aquestes comprovacions per a poder obtenir un rendiment adequat. En aquests casos, hem de pensar alternatives per a controlar les restriccions d'integritat. S'ha de substituir d'alguna manera l'SGBD.

Principalment, podem considerar dues alternatives:

1) La primera és tenir algun tipus de **control extern***.

* Per exemple, algú que comprovi que les dades introduïdes corresponen a la realitat.

2) L'altra possibilitat, que en direm **control diferit**, consisteix a posar en marxa algun procés en les hores de menys activitat** que faci les comprovacions necessàries. Això es podria fer mitjançant SQL/CLI o SQL Host.

** Durant la nit o els caps de setmana.

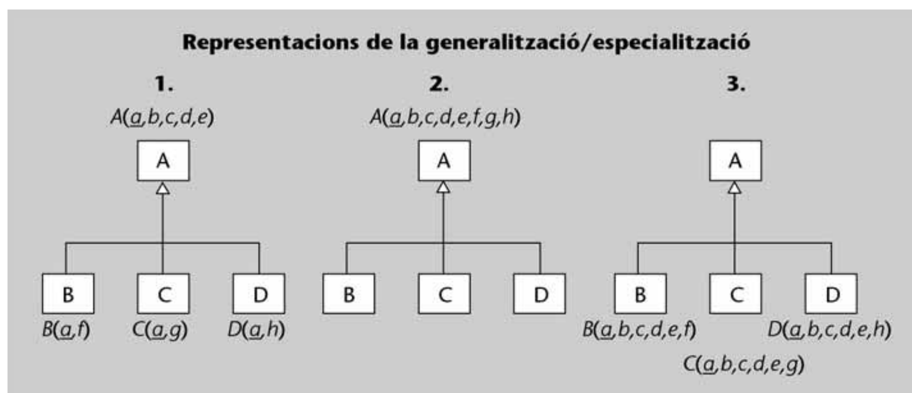
Les **restriccions d'integritat** són imprescindibles i no convé eliminar-les mai d'una BD. Però, per tal de millorar el rendiment del sistema, es poden substituir per altres mecanismes de comprovació.

3. Representació en el model lògic de la generalització/especialització

Un altre problema que cal tenir present en el pas al model relacional clàssic és la representació de diferents classes connectades per relacions de *generalització/especialització*, també conegudes com *superclasse/subclasse*. Quan trobem aquest tipus de relació, la superclasse representa el concepte més general, mentre que les subclasses representen conceptes més específics. Podem trobar diferents tipus de relacions de generalització/especialització atenent a la seva disjuntivitat i completitud.

Tenim una **especialització disjunta** si cap instància de la superclasse està en més d'una subclasse, mentre que tenim una **especialització completa** si tota instància de la superclasse està, almenys, en una de les subclasses.

Hi ha diferents maneres de representar la generalització/especialització en el model relacional:



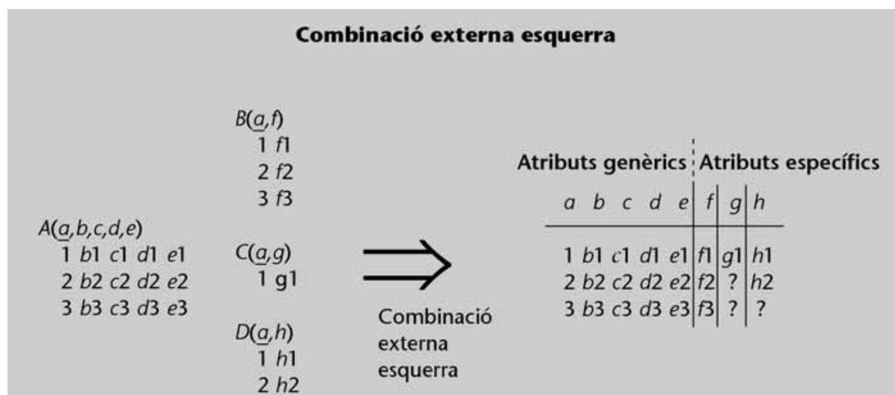
1) Una taula per a cadascuna de les classes, on la taula de la superclasse conté els atributs comuns, i les taules de les subclasses, els específics a cadascuna. Si triem aquesta representació, cal que considerem les complicacions per a consultar la informació. Tinguem en compte que les subclasses hereten els atributs de la superclasse. Aleshores, cada cop que vulguem veure tant els atributs genèrics com els específics de les instàncies d'una subclasse, haurem de fer una combinació (*join*). Si no sabem en quina subclasse hi ha les dades que volem veure o tenim una especialització no disjunta, caldrà fer més d'una combinació alhora (una per cada subclasse), que doni com a resultat una fila per cada tupla de la taula corresponent a la superclasse, independentment del fet que satisfaci la condició de combinació o no.

Justament per a solucionar aquest problema de consulta hi ha una operació de l'àlgebra relacional coneguda com a *combinació externa esquerra**, que posa va-

* En anglès, *left outer join*.

lors nuls a la "part dreta" quan una fila de la taula "esquerra" no compleix la condició de combinació. Aquesta operació està implementada a gairebé tots els SGBD comercials i permet d'obtenir totes les dades que volem amb una única consulta, independentment del tipus d'especialització que tinguem i del fet de saber en quina subclasse es troben les dades o no.

Vegeu la definició de *combinació externa* que s'explica en el mòdul "El model relacional i l'àlgebra relacional" de l'assignatura *Bases de dades I*.



2) Només una taula amb tots els atributs de totes les classes. Aquesta opció només pot ser interessant, depenent del tipus d'especialització i si hi ha moltes subclasses sense atributs. Observem que aquesta representació pot generar molts valors nuls i s'hi perd la visió conceptual en què tenim diferents classes (ara només tenim una taula). Com a cas extrem, si hi ha molts atributs específics en totes les subclasses i tenim una especialització disjunta i incompleta, les files de la nostra taula tindran la majoria de columnes amb valors nuls, ja que utilitzaran com a molt les columnes corresponents a atributs d'una de les subclasses.

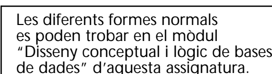
3) Diferents taules per a cada subclasse que guardin també columnes corresponents als atributs de la superclasse. Aquesta darrera opció és aconsellable únicament si tenim una especialització disjunta i completa. Si l'especialització no és disjunta, es duplicaran dades dels atributs genèrics de les instàncies que estiguin en més d'una subclasse. A més, si l'especialització no és completa, és a dir, si hi ha instàncies que no pertanyen a cap de les subclasses, on en guardem les seves dades?

En general, la millor representació d'una generalització/especialització és sempre l'opció **1**, que és la més flexible i propera al disseny conceptual (tenim una taula per cada classe). Només en casos molt concrets pot resultar millor l'opció **2** o la **3**, ja que en general poden provocar l'aparició de molts valors nuls i s'allunyen del que realment volem representar: l'existència de diferents classes.

Hi ha diferents maneres d'emmagatzemar les dades corresponents a classes relacionades per generalització/especialització. Bàsicament, hem de triar entre tenir diverses taules per a les subclasses, una única taula per a la superclasse, o taules diferents per a la superclasse i per cada subclasse. En aquesta tria influirà especialment el fet que l'especialització sigui disjunta o no, i completa o no.

4. Representació d'atributs instanciats en diferents valors

És relativament habitual trobar atributs instanciats en diversos valors alhora. Però recordeu que la primera forma normal (1NF) del model relacional obliga que els atributs siguin atòmics. Per tant, hem de veure què cal fer en aquests casos.



Les diferents formes normals es poden trobar en el mòdul "Disseny conceptual i lògic de bases de dades" d'aquesta assignatura.

Atribut instanciat en diferents valors

Possibles raons per a haver obtingut valors diferents per a un mateix atribut podrien ser, per exemple, l'existència de tres aparells diferents de mesura, o simplement, haver mesurat en tres moments diferents.

Dues solucions a aquest problema són aquestes:

1) Guardar cada valor en una columna diferent de la taula:

$\texttt{Mesura(}\underline{\texttt{clau}}\texttt{,valor}_{\texttt{aparell1}}\texttt{, valor}_{\texttt{aparell2}}\texttt{, ...,valor}_{\texttt{aparelln}}\texttt{)}$

En cas de triar aquesta opció, el nombre de valors de l'atribut no hauria de ser molt gran i hauria d'estar clarament fixat *a priori*. Si un cop tenim la BD en funcionament volem afegir/treure un valor, haurem d'afegir/treure una columna a la taula, amb els problemes que això pot comportar. A més, convé tenir en compte que si no disposem de tots els valors d'una fila determinada, aquesta tindrà columnes amb valors nuls. Si el nombre de valors és molt gran o variable, és millor triar una altra opció.

2) Tenir una fila diferent per cada valor:

$\texttt{Mesura(}\underline{\texttt{clau}}\texttt{, }\underline{\texttt{aparell}}\texttt{, valor)}$

En aquest cas, si un valor no hi és, no cal guardar un valor nul. Simplement, no guardem cap fila per a aquell valor. Tinguem en compte que, en aquest cas, la clau canvia. Per a obtenir un valor determinat, ja no n'hi ha prou de tenir l'identificador de la classe de la qual volem consultar l'atribut, sinó que també cal l'identificador del valor per a triar la fila concreta que volem. La clau de la taula no és tan natural com en l'opció anterior.

Un altre punt que cal tenir en compte són els accessos que farem als valors. En la primera opció, en tenir-los tots en una mateixa fila, només caldrà un accés per a accedir-hi. Per tant, si normalment s'accedeix a tots de cop, això serà molt eficient. Per contra, si es vol accedir només a alguns valors de l'atribut, serà molt millor triar la segona opció, ja que ens permetrà d'obtenir només els valors que interessen.

També és important de veure quin és el tractament que es fa dels valors de l'atribut. Recordem que les funcions d'agregació de l'SQL (*SUM*, *AVG*, etc.) s'apliquen en agrupar files (clàusula *GROUP BY*), i no columnes. Per tant, si els

valors es guarden en columnes diferents, aquestes funcions no es poden aplicar tan fàcilment en les nostres consultes.

Per a tenir un atribut instanciat en diferents valors a una BD relacional, podem triar entre tenir diferents columnes o diferents files per a cada valor. Hi ha alguns factors importants que hem de tenir en compte per a triar l'opció més convenient, com ara el nombre de valors, la variabilitat d'aquest nombre, la unitat d'accés, la clau que volem utilitzar, i la facilitat en el tractament d'agregats.

En la taula següent s'enumeren els factors esmentats:

Factor	Columnes	Files
Nombre de valors	Pocs	Molts
Variabilitat de nombre de valors	Baixa	Alta
Unitat d'accés dels valors	Tots de cop	D'un en un
Característiques de la clau	"Natural"	"Artificial"
Tractament d'agregats amb SQL	Difícil	Fàcil

5. Abraçades mortals de definició i de càrrega

Concretament, en l'àmbit del disseny de BD, ens trobem dos tipus d'abraçada mortal. El primer tipus es pot tenir quan intentem de crear les taules; el segon es dóna en inserir dades. Tots dos es deuen a l'existència de claus foranes creuades.

Abraçada mortal

En general, entenem per *abraçada mortal* (o, en anglès, *deadlock*) una situació en què *A* espera *B*, mentre *B* espera *A*, de manera que cap dels dos actua perquè espera que l'altre ho faci primer.

En el cas de l'**abraçada mortal de definició**, ens trobem que una taula T_1 té una clau forana cap a una taula T_2, i la taula T_2 té una clau forana cap a la taula T_1. Així, no podem crear T_1 fins que no existeixi T_2, ni podem crear T_2 fins que existeixi T_1, perquè cadascuna necessita poder fer referència a l'altra.

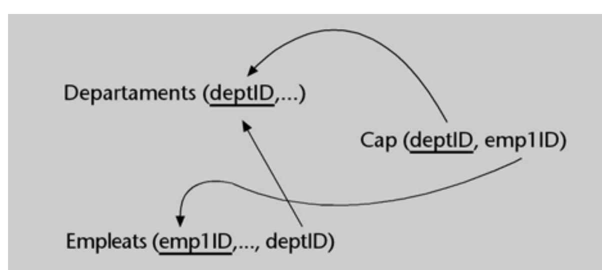
Exemple d'abraçada mortal de definició

Considerem una classe ***Departaments*** i una classe ***Empleats*** de manera que cada empleat pertany a un departament i cada departament té un cap. Atès que no tots els empleats seran cap d'un departament, la taula d'empleats tindrà una clau forana que indicarà quin és el departament de cada empleat i la taula de departaments tindrà una clau forana que apuntarà al cap de cada departament a la taula d'empleats.

La millor manera de solucionar això és crear una tercera taula T_3 de manera que T_1 no faci referència a ningú, T_2 faci referència a T_1 i T_3 faci referència tant a T_1 com a T_2. Amb això guardem la informació que volíem (les dues associacions entre T_1 i T_2), al mateix temps que podem crear les taules començant per T_1 i acabant per T_3. La taula T_1 es pot crear primer perquè no fa referència a ningú. En segon lloc, creem T_2 també sense problemes perquè pot fer referència a T_1, que ja existeix. Finalment, creem T_3, que fa referència a T_1 i T_2, que ja s'han creat amb anterioritat.

Solució per a l'abraçada mortal

La millor solució per a la situació d'abraçada mortal plantejada a l'exemple anterior és la que es veu a continuació:



L'altra solució a aquest problema (proposada als manuals d'algun SGBD i utilitzada per moltes empreses) que pot semblar més simple, però que a la llarga ens continuarà donant problemes és deixar només les dues taules que hi havia inicialment, crear la primera sense clau forana, la segona ja amb clau forana i finalment, modificar la primera (mitjançant la sentència *ALTER TABLE* de l'SQL) i afegir-li la clau forana que falta. D'aquesta manera, també evitem l'abraçada mortal de definició, però ens tornarà a aparèixer quan intentem d'inserir dades.

En aquest darrer cas, ens trobarem amb una **abraçada mortal de càrrega**, ja que tant si inserim primer en una taula, com si ho fem primer a l'altra, violarem una restricció d'integritat.

Exemple d'abraçada mortal de càrrega

Considerem les taules *Departaments* i *Empleats* ja creades, amb les respectives claus foranes entre si. Suposem que volem crear un departament nou amb els seus empleats. Què inserim primer, les dades del departament o les dels empleats? No podem crear un departament, perquè ha de tenir un cap (que ha de ser un empleat) i encara no hem introduït cap empleat. Però, tampoc no podem introduir les dades de cap empleat, perquè necessàriament han de fer referència al seu departament i aquest encara no existeix a la BD.

És important que observem que si per a solucionar l'abraçada mortal de definició hem creat una tercera taula, no tindrem cap problema en la inserció de dades, que es pot fer tranquil·lament en el mateix ordre que hem creat les taules. Si per a poder crear les taules hem utilitzat la sentència *ALTER TABLE*, per a poder fer les insercions haurem de desactivar les RI d'alguna manera.

La primera solució que tenim per a poder fer les insercions, disponible a Oracle 7 i Microsoft SQL Server 7.0, és la desactivació de les RI d'alguna de les taules. Igual que abans, hem utilitzat la sentència *ALTER TABLE* per a afegir l'RI que tancava l'abraçada un cop ja havíem creat les dues taules, ara l'utilitzarem per a treure-la mentre fem les insercions.

```
ALTER TABLE nom_taula DISABLE CONSTRAINT nom_restricció;
```

Cal adonar-se del perill que representa desactivar una RI d'aquesta manera. Mentre aquesta restricció està desactivada, qualsevol procés pot modificar la BD i deixar-la inconsistent sense que ens n'adonem. Una solució molt més bona és la proposada per l'estàndard (semblant a la que ja oferia l'RDB de Digital) i ja disponible tant a Informix com a Oracle 8. Es tracta de no desactivar la restricció, sinó d'ajornar-ne (o diferir-ne) la comprovació fins al final de la transacció. En aquest moment, en executar la clàusula *COMMIT*, ja haurem introduït les dades de les dues taules, i totes les referències haurien de ser correctes.

```
SET CONSTRAINT {ALL | nom_restricció} {INMEDIATE | DEFERRED};
```

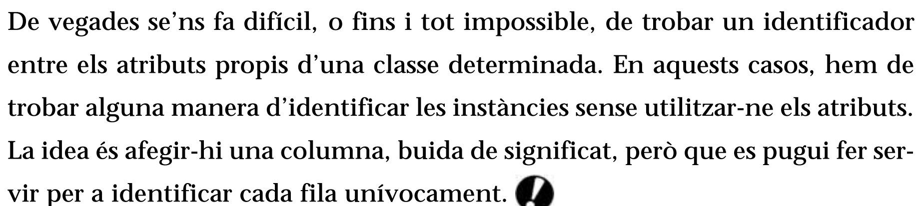
Solució d'alguns SGBD a l'abraçada mortal de càrrega

En l'exemple anterior, en què volíem crear un departament nou amb empleats nous, es podria diferir la comprovació de les RI fins que tinguéssim totes les dades introduïdes. Així, podríem inserir totes les dades tant del departament, com dels empleats dins una transacció sense cap problema perquè al final de la transacció el cap del departament ja estarà introduït (la seva referència serà correcta) i les dades del departament ja estaran introduïdes (la referència de cada empleat al seu departament també serà correcta).

És importantíssim de veure la diferència que hi ha entre desactivar una restricció i diferir-ne la comprovació. En el primer cas, no es comprovarà per a cap operació de cap usuari, mentre que en el segon, sempre es comprovarà per a tothom, només que es farà una mica més tard del que és habitual.

Podem trobar referències creuades entre dues taules, derivades de l'existència de claus foranes entre aquestes, que en dificulten tant la creació com la introducció de dades. La millor solució és evitar que existeixin. Si això no pot ser i resulta que les taules ja estan creades de manera que en fer les insercions de les dades es produeix una abraçada mortal, encara es pot solucionar si la comprovació de les restriccions d'integritat es difereix fins al final de la transacció.

6. Substituts de la clau primària



De vegades se'ns fa difícil, o fins i tot impossible, de trobar un identificador entre els atributs propis d'una classe determinada. En aquests casos, hem de trobar alguna manera d'identificar les instàncies sense utilitzar-ne els atributs. La idea és afegir-hi una columna, buida de significat, però que es pugui fer servir per a identificar cada fila unívocament.

Entenem per *substituts* el que en anglès es diu *surrogates*.

Els **substituts** acostumen a ser comptadors que s'incrementen cada cop que afegim una fila nova a la nostra taula. No estan concebuts per a ser consultats, sinó per a ser comparats amb l'objectiu de determinar la identitat de les instàncies.

La dificultat de trobar identificadors de classes

En un primer moment, podem pensar a identificar les diferents persones pel seu nom i cognoms. Però ens adonarem que no és gaire difícil de trobar persones que es diuen exactament igual (hi ha cognoms massa comuns). La solució a l'Estat espanyol pot semblar clara: utilitzar el DNI. Però malgrat que sembli estrany, el DNI tampoc no identifica les persones. A causa d'errors en el sistema d'assignació de números, hi ha persones que comparteixen el mateix número de DNI.

Podem trobar dos tipus de substituts, segons qui els gestioni, de sistema i d'usuari:

1) E.F. Codd va introduir els **substituts de sistema** en el seu model RM/T ja l'any 1979, com a identificadors generats de manera automàtica per l'SGBD que no es poden repetir dintre d'una mateixa taula. Els sistemes actuals acostumen a permetre valors molt grans per a no restringir el màxim de files que pot tenir la taula. En anglès es coneixen amb el terme *RowID* i es poden consultar normalment amb la sentència *SELECT*, com qualsevol altra columna de la taula.

Amb el pas del temps, els substituts de sistema han evolucionat cap als identificadors d'objecte (*OID*) dels sistemes OO. La diferència fonamental entre els *OID* i els *RowID* és que un *OID* identifica un objecte dins la base de dades, mentre que el *RowID* identifica una fila només dins una taula. És a dir, dues files en taules diferents poden tenir el mateix *RowID*, mentre que dos objectes mai no poden tenir el mateix *OID*. Encara que esborrem un objecte, no se'n reutilitzarà l'*OID*.

Abreugem *identificador d'objecte* amb la sigla *OID*.

2) Respecte als **substituts d'usuari**, a més d'utilitzar-los en taules que no tinguin una clau primària clara (tal com hem vist abans en el cas de les persones), també es poden fer servir quan la clau estigui composta de massa atributs o, simplement, quan sigui massa llarga. Això en facilitarà la referència en cas de tenir claus foranes o índexs.

Quan una taula no té cap conjunt de columnes que n'identifiqui les files, o aquest conjunt és massa gran, podem utilitzar substituts de la clau primària. Aquests substituts no tenen cap significat concret, però ens donen valors diferents per cada tupla o objecte.

Ús de substituts en diferents SGBD

Molts SGBD faciliten la utilització de substituts d'usuari mitjançant sentències (encara no estandarditzades) que els generen automàticament. L'Oracle els anomena *SEQUENCE*; l'Informix, *SERIAL*; i Microsoft SQL Server, *IDENTITY*.

a) Podríem dir que una *SEQUENCE* és un generador de números que podem utilitzar quan i on interessi (per exemple, en les sentències d'inserció per a donar valor a una columna determinada). Ofereix les operacions ***currval***, que dóna el valor en curs, i ***nextval*** que retorna sempre un valor únic encara que diferents usuaris la cridin.

```
CREATE SEQUENCE nom_sequencia [INCREMENT BY n] [START WITH n];
```

b) Per contra, la sentència *SERIAL* d'Informix és simplement un tipus de dades més, que podem utilitzar en la definició de qualsevol columna dins una taula. A partir d'aquest moment, no cal que donem valor a aquesta columna quan fem insercions, perquè prendrà valors automàticament.

```
CREATE TABLE nom_taula(id SERIAL, ... );
```

c) La sentència *IDENTITY* és molt semblant a la *SERIAL* d'Informix. Com aquesta, també es pot utilitzar en crear una taula. Si ho fem, el propi sistema genera, de manera incremental, un valor únic per a la columna que indiquem.

```
CREATE TABLE nom_taula(id INT IDENTITY(inici,increment), ... )
```

7. Freqüència de processos i volums de dades

Ja hem vist com modelitzar conceptes per tal de representar-los en una BD. Però això només és l'inici del procés de disseny. En aquest apartat veurem com és d'important tenir en compte factors com la freqüència de processos i el volum de dades, un cop es té el model conceptual i lògic, per a passar al model físic. Conèixer la quantitat de dades i l'ús que se'n farà permet d'afinar el disseny per a millorar el rendiment de la BD.

7.1. Freqüència de processos

Un punt essencial per a fer un bon disseny d'una BD és considerar per a què s'utilitzarà i com. Cal recordar sempre que les dades es guarden per a ser emprades en algun procés. Hem d'identificar els processos crítics d'entre tots els que accediran a la nostra BD i afavorir-los.

Trobarem dos tipus de processos crítics: els que s'executen molt freqüentment, i els que tenen com a requeriment un temps de resposta màxim determinat. Acostumem a trobar aquests últims en sistemes de temps real i/o en processos que demanen una determinada interactivitat amb l'usuari.

Els sistemes de temps real

Anomenem *sistemes de temps real* aquells sistemes que tenen un temps de resposta clarament afitat per l'activitat en què es fan servir (per exemple, un sistema de control aeri d'un aeroport).

Clarament, si podem triar entre dos dissenys, ens quedarem amb el que millori el temps d'execució dels processos més crítics. Tot i això, de vegades, no n'hi ha prou de triar el disseny més adient per a obtenir el temps de resposta desitjat. En aquest cas, haurem de trobar alguna altra manera de reduir aquest temps.

Les implicacions de la interactivitat amb l'usuari

S'han fet estudis que demostren que una persona es comença a posar neguitosa si l'ordinador triga més de deu segons en respondre, tot i saber que el que ha demanat necessita un temps per a executar-se.

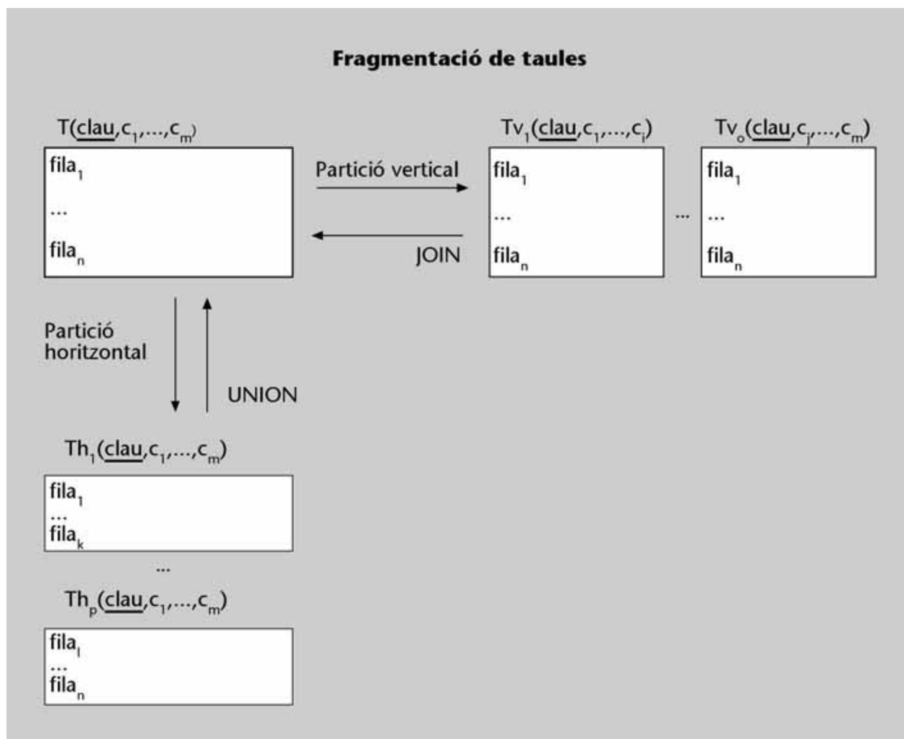
Solucions per a reduir el temps de resposta

Si un procés crític demana una ordenació dels resultats d'una consulta, podem mirar d'implementar o activar algun algorisme especialment ràpid com ara el *quick sort*. També podem millorar el temps de resposta si afegim un índex en una taula determinada, o si canviem el tipus d'índex que hi ha per un de més bo.

En general, cal veure els plans d'execució de les consultes per a trobar quin pas pren més temps i com el podem escurçar.

7.2. Volums de dades (fragmentació de taules)

De la mateixa manera que s'ha de tenir en compte la freqüència amb què s'executen els processos que actuen sobre la BD, també cal considerar quina càrrega comporta cada operació que es du a terme. Així, per a fer un bon disseny, cal veure quin és el volum de dades que hi ha en cada taula.



En cas que una taula sigui massa gran, se'n poden fer particions horitzontals i/o verticals. Tal com podem veure a la figura, fer talls horitzontals implica que hi ha diferents taules amb les mateixes columnes que la taula original, però amb menys files. Per contra, en fer talls verticals, cadascuna de les taules té el mateix nombre de files, però menys columnes. Cal adonar-se del fet que, en fer talls verticals, les columnes que formen la clau primària s'han de repetir a cadascuna de les particions.

En una partició horitzontal, si volem consultar totes les files de la taula original, hem de fer la unió conjuntiva de les consultes sobre cadascuna de les parts. Les particions horitzontals són especialment útils en bases de dades distribuïdes, quan tenim grans volums de dades (en particular, dades històriques), per a permetre el processament en paral·lel i per a facilitar la recuperabilitat.

Vegeu les característiques de les dades històriques en l'apartat 9 d'aquest mòdul.

Exemple d'utilitat de la partició horitzontal de les dades

Una gran empresa d'àmbit estatal podria fer particions horitzontals de les seves dades per províncies, autonomies o regions. Cadascuna de les parts podria estar emmagatzemada físicament en màquines diferents, en llocs diferents, a prop d'on s'hi accedeixi amb més freqüència. A més, les diferents seus podrien col·laborar per a resoldre consultes en paral·lel.

```
                              SELECT *
                              FROM t1
                              UNION
SELECT *          ⇒           ...
FROM t                        UNION
                              SELECT *
                              FROM tn
```

Podem fer particions d'una taula per rangs de valors, segons una condició determinada, o segons algun algorisme de distribució aleatòria que decideixi de manera unívoca en quina part cal posar/buscar cada fila (com ara l'algorisme *Round Robin*). Alguns SGBD comercials disposen de l'expressió *FRAGMENT BY* (o alguna altra d'equivalent) que permet de triar entre aquestes tres maneres de fer particions d'una taula. En qualsevol dels casos, interessa que les parts obtingudes continguin aproximadament el mateix nombre de files per a poder obtenir un bon rendiment del sistema.

Pel que fa a les particions verticals, es fan servir sobretot per a representar especialitzacions d'una classe. En aquest cas, cal combinar-les (fer *joins*) per a resoldre consultes que requereixen atributs guardats en columnes de taules diferents.

La representació de relacions de generalització/especialització s'ha explicat en l'apartat 3 d'aquest mòdul.

Així, doncs, d'una banda, hem d'intentar afavorir l'execució dels processos més crítics mitjançant els millors algorismes disponibles per a afinar el disseny i, d'altra banda, hem d'agrupar files i/o columnes (partició horitzontal i/o vertical) per a dividir les taules massa grans.

8. Redundància de dades: duplicades i derivades

En altres mòduls ja hem explicat el problema que representa tenir redundància a la nostra BD i com la teoria de la normalització ajuda a evitar-la. En aquest apartat veurem que de vegades la redundància pot ser bona o fins i tot necessària, però s'ha de permetre només de manera controlada en els casos que sigui estrictament necessari i documentant-ho de manera adequada perquè en quedi constància.

La teoria de la normalització s'estudia en el mòdul "Disseny conceptual i lògic de bases de dades" d'aquesta assignatura.

8.1. Duplicació de dades

A ser possible, no s'haurien de tenir dades duplicades a la BD. Si normalitzem les dades, no succeirà mai. Això no obstant, la normalització crea moltes taules que després s'hauran de combinar en fer consultes per tal d'obtenir el resultat desitjat. Recordem que la combinació (*join*) és l'operació més costosa d'un SGBD. Si desnormalitzem, dupliquem dades, però evitem combinacions, cosa que pot accelerar considerablement algunes consultes. A més, una consulta expressada amb SQL és molt més llarga i difícil d'entendre quantes més taules impliqui.

Exemple de simplificació de consultes que dupliquen dades

Suposem que tenim una taula amb les dades d'empleats i una altra que conté les dels diferents departaments de l'empresa. Cada cop que vulguem saber en quina ciutat treballa cada empleat, haurem de fer la combinació de les dues taules:

```
SELECT nom_empl, ciutat_dpt
FROM empleats, departaments
WHERE empleats.codi_dpt = departaments.codi_dpt;
```

En canvi, si a la mateixa taula d'empleats guardem la ciutat on hi ha el departament pel qual treballa, estem repetint dades però ens estalviem la combinació i simplifiquem la consulta:

```
SELECT nom_empl, ciutat_dpt
FROM empleats
```

En primer lloc, hem de ser conscients que permetre duplicitat en les dades és malbaratar espai. Haurem de veure si disposem de l'espai de disc necessari per a fer-ho. A més, podria ser que la duplicitat de les dades no reduís el temps de resposta com esperàvem. Si una de les taules ens cap a memòria, la combinació és relativament senzilla (es fa una lectura seqüencial de la taula més gran). En afegir les dades per a evitar la combinació, la taula serà molt més gran, de ma-

Vegeu els diferents algorismes de combinació i selecció en el mòdul "El component de processament de consultes i peticions SQL" d'aquesta assignatura.

nera que si per a fer només una selecció també hem de fer una lectura seqüencial, necessitarem més accessos a disc.

Inconvenients de la desnormalització de dades

Continuem amb l'exemple anterior. Si tenim molts empleats a cada departament i desnormalitzem, guardem molts cops la mateixa informació (amb el consegüent malbaratament d'espai), i fer una lectura seqüencial de la taula d'empleats pot ser significativament més costós.

El més important és que, juntament amb el problema d'espai, tindrem un problema d'actualització. Si algú vol modificar una dada que estigui emmagatzemada a més d'un lloc, ho haurà de fer en cadascuna de les còpies, si no, es generaran inconsistències a la BD.

Exemple de problemes d'actualització per duplicació de dades

Si en l'exemple anterior guardem per cada empleat la ciutat en què treballa i un departament canvia de seu, no sols haurem de modificar la fila corresponent a la taula de departaments, sinó també les dades de cada empleat que treballi en aquest departament.

Sembla clar que s'han de posar tots els mitjans perquè això no pugui passar. El primer que hauríem de fer és documentar aquesta duplicitat perquè tothom en tingués constància.

La duplicació de dades empra més espai del necessari per a emmagatzemar la informació i pot crear problemes d'inconsistències a la BD. Això no obstant, pot servir per a evitar operacions de combinació i fer el contingut de la BD més fàcil de consultar als usuaris. Si ho permetem, és imprescindible de documentar-ho.

Exemple en què podria ser admissible la duplicació de dades

Suposem que utilitzem una BD per a enviar cartes regularment als nostres clients (com per exemple fa un banc). Un bon exemple en què es poden duplicar les dades és el de les adreces i les províncies. Si normalitzem, tindrem dues taules: una amb els carrers i potser els codis postal, i l'altra amb els noms de província. Per a poder enviar una carta a un client (cosa força freqüent), caldrà combinar totes dues taules. Si les dades del carrer i la província es guarden juntes, ens estalviarem aquesta costosa operació i gairebé mai no tindrem problemes d'actualització, ja que no és normal que canviï la distribució provincial. A més, l'espai que ocupa un nom de província sembla menyspreable comparat amb la quantitat de dades que voldrem tenir de cada client.

8.2. Dades derivades

Acabem de veure com la redundància de dades evita temps de càlcul en el cas de la combinació. En general, una manera de reduir el temps de qualsevol operació és mitjançant la redundància de dades. Si s'emmagatzema el resultat d'un càlcul molt costós, s'evita de fer-lo cada cop que l'usuari el demani. A més, encara que el procés no sigui especialment costós, també es poden calcular amb antelació algunes dades derivades, si sabem que l'usuari les demanarà. Però no hem d'oblidar que fer això representa utilitzar més espai del necessari.

Exemple de redundància innecessària

Guardar a la BD el preu unitari, la quantitat d'articles venuts i el muntant total d'una venda és un exemple clar de redundància innecessària, ja que fer una divisió o un producte de números reals no és gaire costós. En canvi, invertir una matriu sí és un càlcul costós. Si es guarda la matriu inversa juntament amb l'original, simplement caldrà llegir-la, en comptes de calcular-la. Però això ocuparà el doble d'espai que guardar només una de les dues matrius.

Un punt molt important que cal considerar per a emmagatzemar dades derivades és la relació entre la freqüència amb què es consulten i la freqüència amb què canvien. Un resultat que canvia molt freqüentment i només es consulta de tant en tant no sembla un bon candidat per a ser emmagatzemat. Si les dades canvien massa freqüentment, encara es poden guardar alguns resultats parcials que siguin més o menys constants, i deixar la resta de càlculs per al moment de la consulta.

En qualsevol cas, el que és més important és veure el temps de resposta que l'usuari demana, considerant la càrrega de la màquina i l'espai d'emmagatzematge de què es disposa. Una manera de reduir el temps de resposta d'un càlcul molt llarg podria ser fer els càlculs quan la màquina té menys processos corrent (possiblement durant la nit), guardar els resultats per a quan l'usuari els demani i esperar que entremig les dades no es modifiquin.

Igual que en el cas de dades duplicades, cal documentar allò que es faci i cal anar amb molt de compte amb les inconsistències que es puguin donar. En una actualització, decidir el moment de recàlcul és especialment delicat, ja que, mentre no ho fem, hi ha una inconsistència, i fer-ho immediatament després de cada actualització pot sobrecarregar el sistema.

La **redundància de dades** pot servir per a reduir el temps de resposta o evitar el recàlcul d'alguns valors. Cal sospesar si és millor malbaratar espai a la nostra base de dades per a tenir dades redundants que, a més, poden crear inconsistències, o haver de fer els càlculs sota demanda, cada vegada que l'usuari ho sol·licita.

9. Històrics

El temps és un atribut universal, present a la gran majoria de sistemes d'informació, i demana una atenció especial. Actualment, els SGBD comercials es limiten a definir tipus de dades específics per a representar el temps, com ara *Time*, *TimeInterval*, o *Date*. El tractament d'aquests tipus d'atributs és encara un tema de recerca.

Els SGBD actuals tracten les dades com una fotografia estàtica de la realitat. El dissenyador de la BD mateix és qui s'ha d'encarregar d'emmagatzemar atributs temporals i dotar-los de semàntica per a donar-hi dinamicitat.

Exemple de tipus d'atributs temporals (I)

Donat un fet com, per exemple, un accident, en podem distingir diferents tipus de temps associats:

- Temps d'ocurrència (quan ha succeït l'accident).
- Temps de comunicació (quan s'ha demanat ajuda).
- Temps d'auxili (quan ha arribat l'ajuda).
- Temps de publicació (quan ha sortit publicat al diari).
- Etc.

En una base de dades distingim principalment tres tipus de temps:

1) Temps vàlid: temps en què alguna cosa és certa en la realitat modelitzada. Serà el mateix a qualsevol BD.
2) Temps de transacció: temps en què una certa BD té constància d'un fet.
3) Temps d'usuari: qualsevol atribut definit sobre un domini de temps i que l'SGBD no interpreta. És l'usuari mateix qui s'encarrega de gestionar-lo.

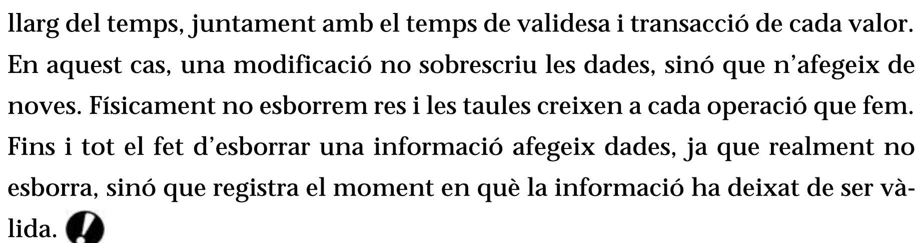
Exemple de tipus d'atributs temporals (II)

Continuant amb l'exemple de l'accident, el temps vàlid seria el moment exacte en què s'ha produït l'accident, mentre que el temps de transacció de la BD seria el moment en què hi introduïm les dades. Un possible temps d'usuari fóra la data de naixement de la persona accidentada.

Fixeu-vos que el temps vàlid de l'accident serà el mateix per a qualsevol BD, mentre que el temps de transacció, no. El temps de transacció de la BD del servei d'ajuda serà previ al temps de transacció de la BD de l'hospital, que serà previ al de la BD del diari.

Un SGBD temporal és aquell que suporta temps vàlid i/o temps de transacció. És a dir, ofereix mecanismes específics per tractar aquests tipus d'atributs.

Si volem reflectir a la BD el pas del temps, no n'hi ha prou de guardar un únic valor per cada dada, sinó que hem d'emmagatzemar tots els valors que pren al



llarg del temps, juntament amb el temps de validesa i transacció de cada valor. En aquest cas, una modificació no sobrescriu les dades, sinó que n'afegeix de noves. Físicament no esborrem res i les taules creixen a cada operació que fem. Fins i tot el fet d'esborrar una informació afegeix dades, ja que realment no esborra, sinó que registra el moment en què la informació ha deixat de ser vàlida.

Per raons legals, acostuma a ser necessari conservar les dades un mínim de quatre o cinc anys. En qualsevol cas, tard o d'hora tindrem problemes d'espai. Si no volem sobrecarregar el nostre SGBD amb el tractament de dades "velles", les haurem de treure del sistema. Per tant, hem de preveure mecanismes de buidat periòdic, com ara passar dades a cinta o fins i tot paper (mai esborrar-les definitivament) quan arriben a una antiguitat determinada.

La necessitat de conservar les dades

Si volem analitzar l'evolució de la nostra empresa, és possible que estiguem interessats a mantenir les dades durant cinc anys o fins i tot més. Darrerament, la informació es considera un bé més de qualsevol empresa (com podria ser la maquinària o els terrenys on està situada); per tant, cal pensar-s'ho dues vegades abans de llençar-la (de la mateixa manera que no llençaríem una màquina quan encara funciona o no vendríem uns terrenys que ens poden fer servei).

Una altra solució per a millorar el rendiment del nostre SGBD és fer particions horitzontals de les taules, de manera que cada període de temps estigui emmagatzemat en una taula diferent. Atès que les consultes acostumen a fer referència a períodes de temps consecutius i relativament curts, una consulta només haurà d'accedir a una de les particions. Tot i que es particionin les taules, arribarà un moment que no podrem evitar haver de treure dades del sistema. Aquesta solució només serveix per a endarrerir aquest moment.

Deixant de banda els problemes d'espai que puguem tenir, també cal tenir en compte que, en períodes de temps llargs, és segur que variaran les RI de les nostres dades. Recordem que una RI s'aplica a totes les files d'una taula i no sols al subconjunt que ens interessa. Per tant, les particions horitzontals no són útils només per a millorar el rendiment, sinó que hauríem de fer particions de les taules segons els períodes de validesa de les RI i definir un conjunt diferent d'RI per cada taula. L'altra opció, sempre desaconsellable, és desactivar la comprovació de les RI conflictives.

Raons per l'aparició de canvis en les RI

Les restriccions d'integritat de la nostra BD es poden veure afectades per canvis en una llei o simplement en la manera com nosaltres mateixos entenem el negoci (el que abans no admetíem, ara ho fem o a l'inrevés).

Resum

En aquest mòdul hem vist diferents temes importants que convé tenir en compte en el disseny d'una base de dades. S'ha presentat un conjunt de possibles problemes que podríem trobar quan dissenyem o administrem un sistema i se n'han proposat algunes solucions.

Els primers apartats han presentat alguns paranys en què podem caure durant la fase de disseny conceptual, juntament amb altres problemes que es donen quan passem al model lògic. En els últims apartats, s'han vist problemes més relacionats amb el disseny físic i l'afinament de la nostra base de dades.

Activitats

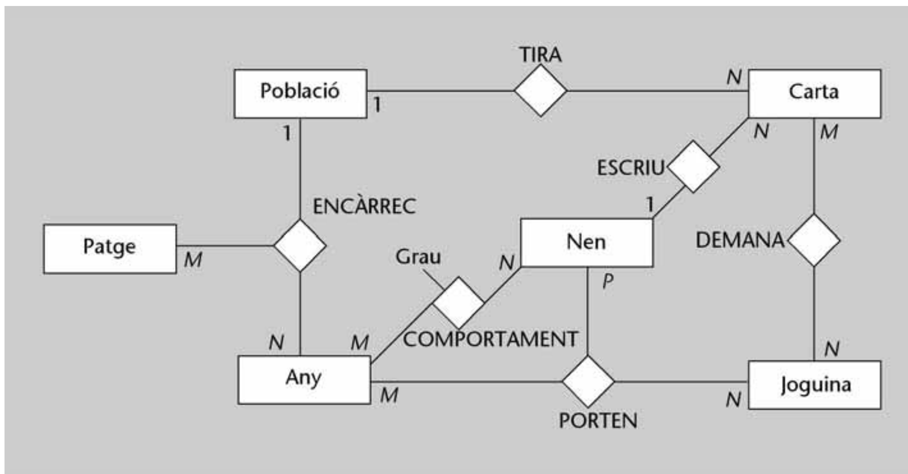
1. Busqueu en la documentació de l'Informix Personal els mecanismes de fragmentació de taules que ofereix.

2. Vegeu a la documentació de l'Informix Personal les diferents opcions que ofereix per a utilitzar substituts.

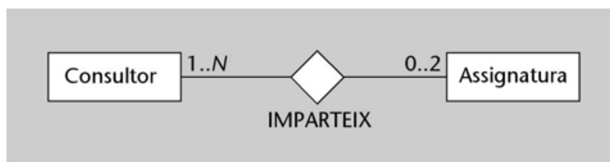
3. Implementeu una generalització/especialització amb l'Informix Personal i practiqueu la combinació externa.

Exercicis d'autoavaluació

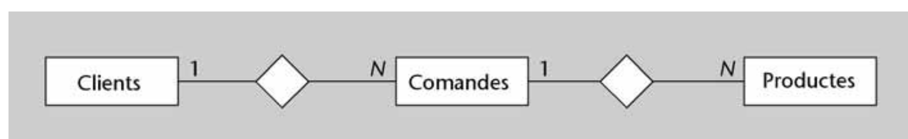
1. Digueu quins problemes trobeu en aquest disseny.



2. Traduïu l'esquema conceptual següent expressat en UML a model relacional i digueu quines restriccions d'integritat s'haurien de comprovar amb SQL Host, SQL/CLI, o SQL/PSM.



3. Com modificaríeu el disseny següent per a poder tenir un històric de comandes?

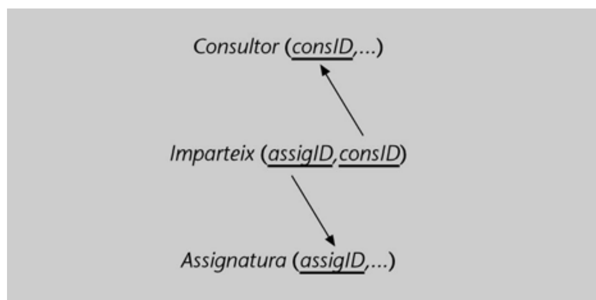


Solucionari

1. En aquest disseny hem caigut en un parany de ventall. Si ens fixem en les classes *Nen*, *Carta* i *Població*, ens adonarem que l'associació que realment ens interessa no és la que hi ha entre *Carta* i *Població*, ja que no podríem decidir quin patge ha de portar els regals als nens que haguessin enviat cartes a poblacions diferents. Hem de tenir una associació entre *Població* i *Nen*, atès que l'important en realitat és saber exactament a quina població viu cada nen.

La multiplicitat 1 de l'associació *Encàrrec* per a *Població* és totalment correcta, atès que un patge, un any concret, només atén una única població. A més, no podem representar *Població* simplement com un atribut perquè té una altra associació.

2. En traduir aquest esquema conceptual a model lògic, obtindríem tres taules, una per cada classe i una altra per l'associació. Aquestes taules estarien relacionades per claus foranes tal com podem veure a la figura següent:



Però només amb això no reflectim totes les restriccions d'integritat que hi havia al model conceptual. Caldran dos programes amb SQL Host, SQL/CLI o SQL/PSM. Un per a comprovar que un consultor té assignades un màxim de dues assignatures, i un altre per a assegurar que cada assignatura té com a mínim un consultor assignat.

3. Un històric es pot dissenyar de manera explícita o implícita. Fer-ho de manera explícita implica tenir una classe *Data* amb la qual associem les altres classes. En aquest cas, això no és necessari, ja que les comandes tindran un atribut que ens dirà a quin dia correspon. Per tant, aquest disseny ja té un històric de comandes implícit.

Glossari

base de dades
Conjunt de dades en què estem interessats.
sigla: BD

BD
Vegeu *base de dades*

combinació
en join
Operació de l'àlgebra relacional que permet de concatenar tuples de dues relacions diferents.

database
Vegeu *base de dades*

entity-relationship model
Vegeu *model entitat-interrelació*

ER
Vegeu *model entitat-interrelació*

identificador de files
en Row Identifier
Codi intern del sistema que identifica cada fila dins una taula d'una base de dades relacional.
sigla: RowID

identificador d'objectes
en Object Identifier
Codi alfanumèric intern del sistema que identifica cada instància dins una base de dades orientada a objectes.
sigla: OID

integrity constraint
Vegeu *restricció d'integritat*

model entitat-interrelació
en entity-relationship model
Model conceptual de dades, dels anomenats semàntics, definit per Peter Chen l'any 1976.
sigla: ER

object oriented
Vegeu *orientació a l'objecte*

OID
Vegeu *identificador d'objectes*

OO
Vegeu *orientació a l'objecte*

orientació a l'objecte
en object oriented
Paradigma de programació i modelització que dóna una importància especial a la correspondència entre les estructures de dades en un sistema informàtic i els conceptes (objectes) del món real.
sigla: OO

restricció d'integritat
en integrity constraint
Regla que han de complir les dades que s'introdueixen en una base de dades. Si la restricció d'integritat no es verifica, les dades són errònies i es parla de base de dades corrupta.
sigla: RI

RI
Vegeu *restricció d'integritat*

RowID
Vegeu *identificador de files*

SGBD
Vegeu *sistema de gestió de bases de dades*

sistema de gestió de bases de dades
en database management system
Programari d'ajuda en la gestió de les bases de dades.
sigla: SGBD

UML
en *unified modeling language*
Llenguatge visual de modelització concebut per a ser utilitzat en tot el procés d'enginyeria del programari. Una part d'aquest llenguatge està dedicada a la modelització conceptual de les dades.

Bibliografia

Batini, C.; Ceri, S.; Navathe, S.B. (1992). *Conceptual Database Design: An Entity-Relationship Approach*. Reading: Addison-Westley.

Hansen, G.W.; Hansen, J.V. (1997). *Diseño y Administración de bases de datos* (2a. ed.). Prentice Hall.

Howe, D.R. (1989). *Data Analysis for Data Base Design* (2a. ed.). Edward Arnold.

Teorey, T.J. (1994). *Database Modeling & Design. The Fundamental Principles* (2a. ed.). San Francisco: Morgan Kaufmann Publishers.